

# Phase-Sensitive Detection in the undergraduate lab using a low-cost microcontroller

K. D. Schultz\*

*Hartwick College, Oneonta NY 13820 USA*

(Dated: January 27, 2015)

Phase-sensitive detection (PSD) is an important experimental technique that allows signals to be extracted from noisy data. PSD is also used in modulation spectroscopy and is used in the stabilization of optical sources. Commercial lock-in amplifiers that use PSD are often expensive and host a bewildering array of controls that may intimidate a novice user. Low-cost microcontrollers such as the Arduino family of devices seem like a good match for learning about PSD; however, making a self-contained device (reference signal, voltage input, mixing, filtering, and display) is difficult, but in the end the project teaches students “tricks” to turn the Arduino into a true scientific instrument.

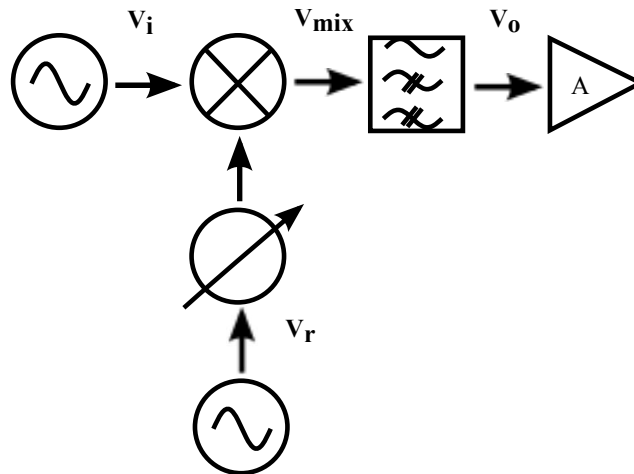


FIG. 1. Block diagram of a traditional PSD.  $V_i$  and  $V_r$  are the input and reference voltages respectively. *LPF* is a low-pass filter and *A* is an amplifier. A phase-shifter is shown after  $V_r$ , but this and the amplifier are not used in this project. See the text for more details.

## I. INTRODUCTION

Lock-in amplification and phase-sensitive detection are important techniques in experimental physics, see<sup>1–3</sup> for pedagogical uses of these techniques. Commercial devices are expensive and can be intimidating for new users. Building a home-made instrument can be instructive<sup>4</sup>; however, doing so requires advanced electronics skills that a student may not already have, putting the emphasis on the electronics and not the method. Conversely, it is possible to perform the mixing and filtering on a computer using a computer's sound card or other low-cost data acquisition devices to handle the input and output, see for instance<sup>5</sup>, but in the author's experience computer driver issues often get in the way of relatively easy implementation.

In this paper I describe a PSD that uses the popular Arduino microcontroller and the Processing programming environment<sup>6</sup>. The major design goal was to make the device as self-contained as possible, a task made difficult by the memory and hardware constraints of a typical microcontroller. While these devices are fantastic for controlling robots and basic data-logging, turning them into scientific instruments requires techniques that go beyond what is normally found in the literature. An added benefit is that these techniques can be used to build other types of instrumentation such as function generators and fast DMMs out of a low-cost microcontroller that typically only cost a few tens of dollars.

## II. ARDUINO

Arduino<sup>7,8</sup> is a catch-all term for a family of open-source hardware based on Atmel micro-controllers with a pre-loaded bootloader so that instead of programming the Arduino with the more difficult, but more powerful, AVR instruction set, users can program with a C++-like language. Programming and communication can be done via USB or through a set of on-board communication pins. There have been sixteen different Arduino-labeled boards produced to date, each of which has its own unique hardware and memory specifications. In this paper, the Uno R3 is used, simply because it was what was on hand and is one of the cheaper and more basic Arduinos. The heart of the Uno R3 is the Atmel ATmega328 microcontroller<sup>9</sup>. The Uno R3 has 14 digital input/output pins, six of which provide Pulse Width Modulation (PWM) output. It also has six analog inputs, and an on-board 16MHz oscillator. The ATmega 328 has 32kB of Flash memory to store programs, and 2kB of SRAM for variable storage. The program given in this paper is easily stored in the Flash memory, but the limited SRAM places severe restrictions on the amount of data that can be taken and manipulated on-board the Arduino.

## III. PHASE SENSITIVE DETECTION

A block diagram for phase-sensitive detection is shown in Fig. 1. The input stage of a PSD consists of two signals, the input of interest and the reference signal. Typically, these two signals are oscillating at the same frequency, but

in the derivations to follow this will only be enforced at the end. After the signals are acquired, the input signal may be amplified and filtered. After the input stage comes the mixing stage where the input and reference signals are combined. There are a number of ways of doing this, but in this implementation the signals will simply be multiplied. Finally, the output of the mixing stage is heavily filtered by a low-pass filter. Once again this can be done using analog circuitry, but here it will be done mathematically.

### A. Generic Phase Sensitive Detection

Let us assume that the input signal  $V_i$  and the reference  $V_r$  are given by

$$\begin{aligned} V_i &= V_1 + V_1 \sin(\omega_s t - \phi_s), \\ V_r &= V_2 \sin(\omega_r t - \phi_r), \end{aligned} \quad (1)$$

A DC offset is included with  $V_i$  because Arduino, like most microcontrollers, only input or output positive voltage; therefore the Arduino outputs an offset reference voltage. This offset is removed during mixing, so  $V_r$  is centered on zero in what follows. Upon multiplication of the two signals in Eq. 1, and making use of a trig identity, the output of the mixing stage  $V_{mix}$  is

$$V_{mix} = V_1 V_2 \sin(\omega_2 t - \phi_2) + \frac{V_1 V_2}{2} \{ \cos[(\omega_2 - \omega_1)t - (\phi_2 - \phi_1)] - \cos[(\omega_2 + \omega_1)t - (\phi_2 + \phi_1)] \} \quad (2)$$

At this stage Eq. 2 is nothing more than what we find in heterodyne detection in radio and optical engineering. The power of coherent detection is that small signals get amplified by a larger local oscillator signal is evident in Eq. 2. Upon mixing of the two signals there are contributions to the signal at the original frequencies and at the sum and difference frequencies. This is the heart of PSD; the output filter is set to reject all frequencies other than the difference frequency of the inputs. Furthermore, if  $V_i$  and  $V_r$  are made to oscillate at the same frequency before entering the PSD, the output  $V_o$  of the PSD simply depends on the phase difference between the two signals

$$V_o = \frac{V_1 V_2}{2} \cos(\phi_o - \phi_1). \quad (3)$$

The final effect of this filtering is to move our output from  $\omega_{1,2}$  to DC. The stronger the filtering, the more noise is rejected and the signal-to-noise increases. However, in effect, the PSD is performing signal averaging, so with each factor of two increase in the signal-to-noise, the collection time increases by a factor of four. The other disadvantage of PSD is that by making the measurements at DC we are placing our signal where  $1/f$  noise dominates<sup>10</sup>. Once again we can see similarities with homodyne detection, and even though we have added no active amplification, the presence of a strong local oscillator can boost a weak experimental signal.

## IV. IMPLEMENTATION

As stated in Sec. I, the ultimate goal of this project was to make the project as self-contained as possible. It would need only a computer for display, the Arduino, and as few passive circuit components as possible. These turned out to be fairly severe constraints and required some exploits that are not commonly presented in introductions to microcontrollers. The structure of this section is to discuss the implementation of each of the sub-systems shown in Fig. 1. Briefly, and following Fig. 2, the Arduino synthesizes a sine wave output from a wavetable. As the output is updated from the wavetable, the input signal is quickly read. Finally after the Arduino has cycled through the complete wavetable, mixing is performed mathematically onboard the Arduino, and the resulting waveforms are sent to the host computer running Processing for filtering and display.

### A. Creating a Reference Signal

Creating the reference signal is in many ways the most significant constraint on the project. The Arduino does not have a true analog output. The only way to approximate an analog signal is to use PWM with a low pass filter circuit on the output to smooth things out. Additional tricks are needed to generate something resembling a sine wave.

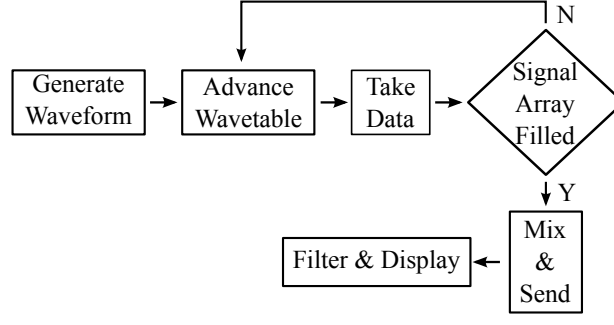


FIG. 2. All the steps through the mixing of the signals are done on the Arduino. The filtering and displaying are done in a Processing sketch.

To generate a reference signal under these limitations a technique sometimes called “bit-banging”<sup>11</sup> is employed. Bit-banging takes advantage of the timers on the Arduino and PWM. In PWM, the duty-cycle (ratio of ‘on’ to ‘off’ times) of a square wave is modulated. The greater the duty-cycle, the longer the PWM pin on the Arduino is held HIGH. Integrating that square-wave produces a DC voltage that increases with the duty-cycle. The final step is to rapidly change this DC voltage so that the desired waveform, in this case a sine-wave, is synthesized.

This paper will only briefly sketch the relevant idea. For details, refer to the comments in the source code for this project, the “bit-banging” paper<sup>11</sup>, or the ATmega328 datasheet<sup>9</sup>. This technique relies on two counters on the Arduino. Timer 1, TCNT1, is a 16-bit timer/counter that is configured as an 8-bit counter. TCNT1 runs at the full Arduino clock frequency, 16MHz, and produces the PWM output signal. Timer 2, TCNT2, runs slower than TCNT1 by a factor of eight, and is used to step through the pre-generated wavetable.

Output Compare Registers (OCRnA) set a flag when the value of TCNTn matches OCRnA. When the OCRnA flag is set each timer can exhibit different behavior depending on how the Arduino is programmed. When the OCR1AL flag is set, the PWM pin is made to go low, but TCNT1 keeps incrementing until it overflows. At which point TCNT1 starts counting again, the PWM pin goes high again, and OCR1LA is reset. So changing the value of OCR1AL determines the duty-cycle of the PWM signal and consequently the DC voltage that is used to construct  $V_r$ . The pre-programmed wavetable for  $V_r$  is what updates OCR1AL. To do this TCNT2, running eight times slower than TCNT1, updates OCR1AL with the next value from the wavetable upon reaching OCR2A. OCR2A is pre-set in the Arduino in the code such that the frequency of  $V_r$  is given by:

$$f_r = \frac{\text{rate of TCNT2}}{\text{OCR2A} \times \text{wavetable length}}. \quad (4)$$

When TCNT2 reaches OCR2A, an interrupt is called which does two things, update OCR1AL to the next value of the wavetable and get an input voltage for the ADC. The last step in generating  $V_r$  is to place a simple low-pass RC filter on the PWM pin to get a smooth sine-wave.

## B. Getting the signal in

The Arduino ADC is a 10-bit successive approximation circuit connected to an 8-channel multiplexer. The measurements are single-ended and normally referenced to a 1.1V on-board reference voltage, although it is possible to use an external reference voltage. Because interrupts are used to create an analog output for the reference signal, it is important to ensure that anything that happens during the interrupt is quick, which is why in Fig. 2, the data is sent for filtering after the wavetable has been completely cycled through. Serial calls are slow.

According to ATMEL the ultimate sampling rate for single-ended measurements is limited by the ADC clock speed. The ADC clock speed is derived from the main system clock, and for maximum resolution should be between 50kHz and 200kHz, however, for purposes of this project satisfactory resolution and accuracy are obtained by setting the ADC clock to a speed of 1MHz. The successive approximation circuit requires 13 clock cycles, giving a sampling rate of 77kHz<sup>13</sup>, which is much faster than this project needs.

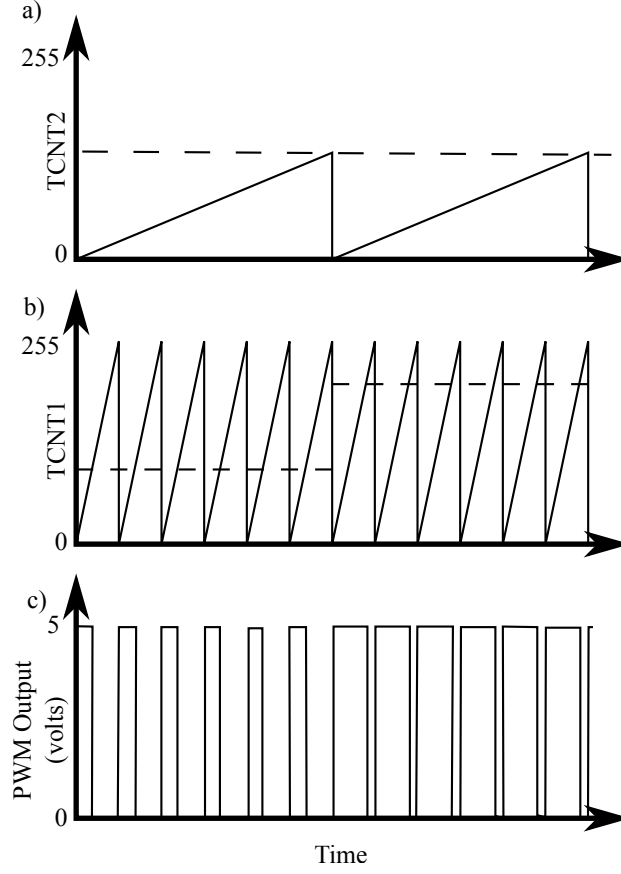


FIG. 3. The “bit-banging” technique requires a slow counter a), a fast counter b), and a pulse-width modulated signal c). The dashed line in a) represents OCR2A, the counter value that calls the necessary interrupt to update OCR1A and read in the input voltage for mixing. The dashed line b) represents the changing OCR1A, which sets the duty-cycle for the PWM. The reference signal that modulates the experiment is the output of a low-pass RC-circuit with c) as the input.

### C. Phase shifts and mixing

Since the values of the reference signal are pre loaded into the Arduino memory, changing the phase while data is being taken is difficult. Often, however, the ability to shift the relative phase between  $V_i$  and  $V_r$  is need to maximize the signal. Additionally, sometimes what is needed in a measurement is the phase shift between the reference and input signals. To meet these needs a cue is taken from dual-phase, lock-in amplifiers<sup>14</sup> and in addition to our reference waveform a quadrature signal, which is just the reference waveform shifted by  $90^\circ$ , is generated. Both the in-phase,  $I = V_{in} \cos(\omega_r t)$  and quadrature  $Q = V_{in} \sin(\omega_r t)$  mixed waveforms are used so that the phase ambiguity is removed by calculating the magnitude signal  $R = \sqrt{I^2 + Q^2}$  and the phase difference between the input and reference signals is  $\tan \phi = Q/I$ . The  $I$  and  $Q$  signals are what are sent from the Arduino to the host computer for filtering and display.

### D. Filtering and display

Once the Arduino finishes running through the waveform, it takes the collected data and sends it via USB to a computer running the Processing IDE<sup>15,16</sup>. Processing is a programming language and development environment that was designed to make it easier for the arts community to become software literate. Like the Arduino, it has a vibrant community that has produced numerous tutorials, examples, and books that make learning Processing relatively easy. Processing is also a direct forebear of the Arduino development system and therefore makes it a natural fit with the Arduino side of this project.

In this project the Processing sketch (Processing-talk for program) initiates serial communications with the Arduino and once communications are established, the Arduino sends the  $X$  and  $Y$  data to the Processing sketch. The Processing sketch applies a recursive, single-pole, low-pass filter?? to the array that is to be displayed  $(I, Q, R, \phi)$ .

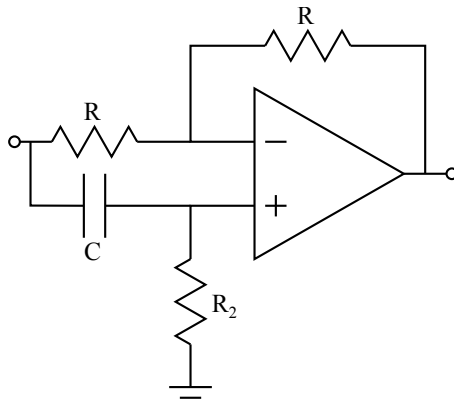


FIG. 4. Phase shifting circuit for testing the Arduino PSD. The op-amp used was a standard 741. Changing  $R_2$  shifts the phase through 180 degrees.

In general a recursive filter is given by

$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + \dots \quad (5)$$

$$+ b_1y[n-1] + b_2y[n-2] + \dots, \quad (6)$$

where  $y$  is the output of the filter and  $x$  is the input data. For this project a single-pole filter was used, so the only coefficients used are given by a single parameter  $x_{\text{decay}}$

$$a_0 = 1 - x_{\text{decay}}$$

$$b_1 = x_{\text{decay}}$$

$$x = e^{-2\pi f_c}, \quad (7)$$

where  $f_c$  is the time constant of the filter. Mathematically this filter is identical to a single-pole RC-filter in electronics. The stronger the filtering, the better it is for lock-in detection, since inadequate filtering causes the output to oscillate at the reference frequency. Ref.<sup>17</sup> has algorithms for calculating coefficients for higher-order filters with faster roll-off, single-pole filters are limited to attenuations of 6dB/decade.

## V. TESTING

To test this project a all-pass phase-shifter was built. Fig. V shows the circuit used to test the Arduino. The voltage gain for this circuit is 1V/V, so the amplitude of the output is unchanged with respect to the input voltage. The high-pass RC filter at the non-inverting terminal of the op-amp controls the amount of phase-shift at the output. At  $\omega_0 = 1/RC$  the phase-shift is  $90^\circ$  and changes by  $90^\circ/\text{decade}$ . By varying  $R_2$ , the  $90^\circ$  point shifts and moves our operating frequency along the phase plot. To test the Arduino PSD, the reference signal generated by the Arduino is sent to the input of the phase-shifting circuit. The output of the phase-shifting circuit becomes the input signal to the PSD. Fig. V shows the  $I$  signal, as displayed in the Processing sketch, as  $R_2$  is varied.

## VI. CONCLUSIONS

A phase-sensitive detector using an Arduino microcontroller has been described. The main design goal of the project was to make the PSD as self-contained as possible, namely that an external reference signal was not needed and that any phase corrections were unnecessary. Aside from the Arduino and a computer the only other hardware needed are a resistor and a capacitor. Furthermore the techniques used for this project can be used in other projects involving using the Arduino microcontroller as a standalone, low-cost, scientific instrument. The “bit-banging” technique is not limited to sine-waves, but can be used to output any synthesized waveform that may be required. More complicated waveforms can be created outside of the Arduino environment and loaded into the EEPROM allowing faster execution and freeing up regular memory.<sup>11</sup>

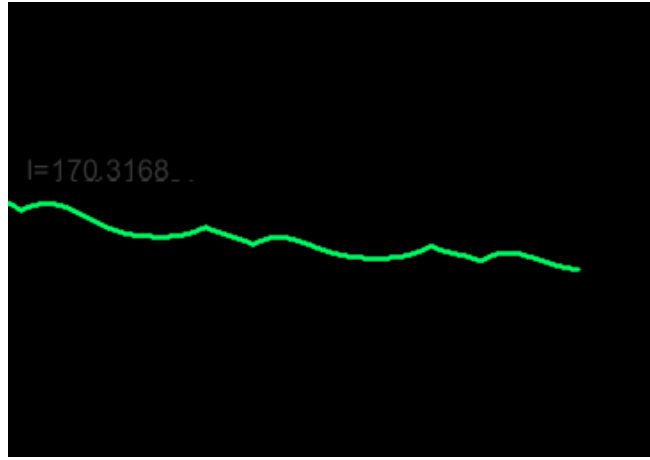


FIG. 5. The in-phase signal of the Arduino PSD as  $R_2$  of the phase-shift test circuit is varied.

### ACKNOWLEDGMENTS

I would like to thank Prof. Lawrence Nienart of Hartwick College for his advice and encouragement throughout this project.

- 
- \* schultzk@hartwick.edu; [www.HartwickChaosLab.github.io](http://www.HartwickChaosLab.github.io)
- <sup>1</sup> W. Yang, “Teaching phase-sensitive demodulation for signal conditioning to undergraduate students”, *Am. J. Phys.* **78**, p. 909–915 (2010).
  - <sup>2</sup> K. G. Libbrecht, E. D. Black, and C. M. Hirata, “A basic lock-in amplifier experiment for the undergraduate laboratory”, *Am. J. Phys.* **71**, p. 1208–1213 (2003).
  - <sup>3</sup> R. Wolfson, “The lock-in amplifier: A student experiment”, *Am. J. Phys.* , **59**, p. 569–572 (1991).
  - <sup>4</sup> P. Horowitz and W. Hill *The Art of Electronics*, 2nd ed (Cambridge Univ. Press, Boston, 1989).
  - <sup>5</sup> M. González, G. Santiago, V. Slezak, and A. “Simple synchronic detection at audio frequencies through a PC sound card”, *Rev. Sci. Inst.* **78**, p. 055108 1–4 (2007).
  - <sup>6</sup> The Arduino and Processing code can be found on my github page, which can be forked and modified. [github.com/HartwickChaosLab/Arduino-Phase-Sensitive-Detector](https://github.com/HartwickChaosLab/Arduino-Phase-Sensitive-Detector)
  - <sup>7</sup> Arduino Project Web Site, <http://www.arduino.cc>
  - <sup>8</sup> M. Margolis, *Arduino Cookbook* 2nd ed., (O’Reilly Media, Sebastopol, 2011).
  - <sup>9</sup> AtMega Data Sheet <http://www.atmel.com/devices/atmega328.aspx>, accessed 1/10/2015.
  - <sup>10</sup> Philip C. D. Hobbs, *Building Electro-Optical Systems: Making It All Work*, (John Wiley & Sons, New York, 2000).
  - <sup>11</sup> J. Thompson, “Advanced Arduino Sound Synthesis”, *Make*, **35**, p. 80–88 (2014).
  - <sup>12</sup> In what follows I will use the convention that “n” at the end of a name represents a generic counter, and if it is replaced by a number that will designate a specific counter.
  - <sup>13</sup> Atmel white paper:AVR20, “Characterization and Calibration of the ADC on an AVR” <http://www.atmel.com/images/doc2559.pdf> accessed 1/10/2015.
  - <sup>14</sup> M. Meade, “Advances in lock-in amplifiers”, *J. Phys. E.* **15**, p. 395–405 (1982).
  - <sup>15</sup> Processing Project Web Site, <http://www.processing.org>
  - <sup>16</sup> C. Reas and B. Fry, *Processing: A Programming Handbook for Visual Designers and Artists*, 2nd ed, (MIT Press, Cambridge, 2014).
  - <sup>17</sup> Steven W. Smith, *The Scientist and Engineer’s Guide to Digital Signal Processing*, (California Tech. Pub. San Diego, 1997).